Bag of Tricks for Node Classification with Graph Neural Networks

Yangkun Wang[1*], Jiarui Jin[1*], Weinan Zhang[1], Yong Yu[1], Zheng Zhang[2], David Wipf[2]
[1]Shanghai Jiao Tong University, [2]AWS Shanghai AI Lab
espylapiza@gmail.com,{jinjiarui97,wnzhang,yyu}@sjtu.edu.cn,{zhaz,daviwipf}@amazon.com

ABSTRACT

Over the past few years, graph neural networks (GNN) and label propagation-based methods have made significant progress in addressing node classification tasks on graphs. However, in addition to their reliance on elaborate architectures and algorithms, there are several key technical details that are frequently overlooked, and yet nonetheless can play a vital role in achieving satisfactory performance. In this paper, we first summarize a series of existing tricks-of-the-trade, and then propose several new ones related to label usage, loss function formulation, and model design that can significantly improve various GNN architectures. We empirically evaluate their impact on final node classification accuracy by conducting ablation studies and demonstrate consistently-improved performance, often to an extent that outweighs the gains from more dramatic changes in the underlying GNN architecture. Notably, many of the top-ranked models on the Open Graph Benchmark (OGB) leaderboard benefit from our techniques.

KEYWORDS

Graph Neural Networks, Node Classification, OGB Leaderboard

1 INTRODUCTION

Recently, machine learning tasks involving graphs have received increasing attention, among which node classification is one of the most prominent examples. Since the remarkable success of graph convolution networks (GCN) [6], many high-performance GNN designs have been proposed to address the node classification problem, such as graph attention networks (GAT) [18] and GraphSAGE [3]. At the same time, we have witnessed a steady improvement in model accuracy as demonstrated on the Open Graph Benchmark (OGB) leaderboard [4]. For example, the top-1 test accuracy for node classification on the ogbn-arxiv dataset has improved from 70.1% (based on node2vec [2]) to 74.1% (based on GAT).

However, these advances are not derived exclusively from the development of model architectures. Refinements including data processing, loss function design and negative sampling also play a major role. Specifically, for semi-supervised learning, it is worthwhile to explore the effective use of the information contained in node features and/or labels. In this context, common approach is to train GNN models that make predictions based on node features and model parameters; however, this strategy cannot directly utilize existing label information (beyond their influence on model parameters through training). In contrast, label propagation algorithms (LPA) [24] spread label information to make predictions, but cannot exploit node features. Although many recent attempts [5, 7] propose to integrate node features and label information by combining GNN and LPA, these approaches suffer from the inherent limitation that LPA requires neighboring nodes to share similar labels and cannot be applied to graphs with edge features.

In this paper, we propose a series of novel techniques covering both label usage and architecture design. Specifically, we first develop a masking technique that enables GNNs to leverage the *original label* as a model input. Based on this, we also design an iterative enhancement which utilizes the *predicted labels* from the previous iteration as input for further training. Additionally, we propose a robust loss function and discuss some details of GCN and GAT designs. We evaluate these modifications and tricks on multiple GNN architectures and datasets, demonstrating that they often lead to significant improvement in node classification accuracy.

Notably, 8 of the top 10 models on the ogbn-arxiv leaderboard, including AGDN [17], C&S [5], FLAG [8] and UniMP [16], benefit from our methods or small variations thereof. Moreover, on the more challenging ogbn-proteins dataset, we can obtain an ROC-AUC of 0.8765, which as of the time of this submission, outperforms all the methods on the OGB leaderboard.

2 PRELIMINARY

2.1 Problem Formulation

Notations. Given a graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of nodes and E is the set of edges, we denote $\boldsymbol{A}$ as the adjacency matrix and $\boldsymbol{D}$ as the diagonal degree matrix. Let $\mathcal{X}$ be the feature space and $\mathcal{Y}$ be the label space, such that we have node features $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)^T$ and labels $\boldsymbol{Y} = (y_1, \ldots, y_N)^T$, with each node associated with a feature vector $\boldsymbol{x}_i \in \mathcal{X}$, while only the first M nodes have available labels $y_1, y_2, \ldots, y_M$. For the multi-class setting, y_i must be in the range from 1 to C, with C being the number of classes; for the two-class setting, y_i must be either +1 or −1. For each dataset $\mathcal{D} = \{(v_i, \boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ (where $\{y_j\}_{j=M+1}^{N}$ is unknown) associated with a graph G, we have the training set $\mathcal{D}_{train}$ ($|\mathcal{D}_{train}| = M$) and the test set $\mathcal{D}_{test}$. The goal of the node classification task is to predict the labels of unlabeled nodes. Given the loss function $\ell(\hat{y}, y)$, our optimization objective is to minimize

*Work done during internship at AWS Shanghai AI Lab.

Woodstock'21, August 14 - 18, 2021, Virtual Conference

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

the aggregated cost $\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{M} \ell(\hat{y}_i, y_i)$, where $\hat{y}$ indicates the predicted label and $\boldsymbol{\theta}$ indicates the parameters.

Label Propagation Algorithm. LPA is a semi-supervised algorithm that assigns labels to unlabeled nodes by propagating observed labels across the edges of the graph, with the underlying assumption that two nodes connected by an edge in the graph are likely to have the same label. Let $\boldsymbol{S} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ be the symmetric normalized adjacency, LPA solves a linear system $\boldsymbol{Y}^* = (1-\lambda)(\boldsymbol{I} - \lambda\boldsymbol{S})^{-1}\boldsymbol{Y}$ by iteratively computing $\boldsymbol{Y}^{(k+1)} = \lambda\boldsymbol{S}\boldsymbol{Y}^{(k)} + (1-\lambda)\boldsymbol{Y}^{(0)}$, where $\boldsymbol{Y}^{(0)} = \boldsymbol{Y}$.

Graph Neural Networks. GNNs are a series of multi-layer feed-forward neural networks that propagate and transform layer-wise features. Among these models, a GCNs architecture is widely employed, which relies on the layer-wise propagation rule

$$\boldsymbol{X}^{(l+1)} = \sigma(\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{X}^{(l)}\boldsymbol{W}^{(l)}), \tag{1}$$

where $\boldsymbol{W}^{(l)}$ denotes a trainable weight matrix of the l-th layer, $\sigma(\cdot)$ is an activation function, and $\boldsymbol{X}^{(l)}$ represents the l-th layer node representation. GAT further leverages masked self-attentional layers to implicitly assign different weights to different neighboring nodes. Let $(v_i, v_j) \in E$ be an edge, then the layer-wise propagation rule of GAT is as follows:

$$\begin{aligned}
\alpha_{ij}^{(l)} &= \frac{\exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T[\boldsymbol{W}^{(l)}\boldsymbol{x}_i^{(l)} \parallel \boldsymbol{W}^{(l)}\boldsymbol{x}_j^{(l)}]\right)\right)}{\sum_{r\in\mathcal{N}(v_i)}\exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T[\boldsymbol{W}^{(l)}\boldsymbol{x}_i^{(l)} \parallel \boldsymbol{W}^{(l)}\boldsymbol{x}_r^{(l)}]\right)\right)}, \\
\boldsymbol{x}_i^{(l+1)} &= \sigma\left(\sum_{v_j\in\mathcal{N}(v_i)} \alpha_{ij}^{(l)}\boldsymbol{W}^{(k)}\boldsymbol{x}_j^{(l)}\right),
\end{aligned} \tag{2}$$

where $\boldsymbol{a}$ is a weight vector of the single-layer feed-forward neural network, and $\parallel$ denotes the concatenation operation.

Combination of Label and Feature Usages. Since both LPA based on label usage and GNN architectures developed on feature usage have achieved good performance, recent attempts [5, 7] have been made to combine them. However, since GNN is a parametric model and LPA is a nonparametric one, and LPA relies on the assumption that adjacent nodes tend to have similar labels, simple combinations may lead to suboptimal solutions, which in part shows the gap in their representation and generalization capabilities. Instead, we propose in Section to exploit the label information by treating random subsets of training labels as input and predicting the remaining ones. We also design a novel label reuse strategy on graphs, which utilizes not only the true label values on training nodes, but also predicted values on test nodes as inputs.

2.2 Existing Tricks

In addition to the combined use of the labels and features techniques, a significant amount of previous work has focused on data processing and usage (e.g., data augmentation, data sampling, normalization), which can also improve model performance.

Sampling. Sampling techniques [1, 3, 25] are often essential to the efficient training of GNNs. For example, recent methods such as FastGCN [1] and LADIES [25] investigate layer-wise and layer-dependent importance sampling to improve the efficiency and effectiveness of GNNs. Negative sampling strategies can also play an important role [21], having been first proposed to accelerate skip-gram training in word2vec [12] and now widely adopted in web-scale graph mining such as PinSAGE [22].

Data Augmentation. For semi-supervised node classification tasks on graphs, overfitting and over-smoothing are two main obstacles when training GNNs. In order to surmount these obstacles, the DropEdge method [13] randomly removes a certain number of edges from the input graph, acting like a data augmenter and also a message passing reducer. In addition to modifying graph structures, another direction inspired by the recent success of adopting adversarial training in computer vision [20], works purely in the node feature space by adding gradient-based adversarial perturbations to the input features, while keeping graph structures unchanged [8].

Renormalization. The renormalization trick was introduced in GCN [6] to alleviate the numerical instabilities and gradient explosion brought about by repeated application of Eq. (1) during the training of deep GCN models. Specifically, we replace $\boldsymbol{A}$ and $\boldsymbol{D}$ with $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{D}}$, where $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ and $\tilde{\boldsymbol{D}} = \boldsymbol{D} + \boldsymbol{I}$. Since $\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}} = \boldsymbol{I}_N + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$, it works similarly to adding a self-loop to each node of the graph.

GCN with Residual Connections. A primitive form of GCN with a linear connection added to the message passing formulation was introduced [6], although it was not originally discussed in combination with the renormalization trick. Subsequently, there has also been a body of work using broader forms of residual connections [10, 14]. One variant we find to be stable and robust directly combines the primitive form of GCN with the renormalization trick:

$$\boldsymbol{X}^{(l+1)} = \sigma\left(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{X}^{(l)}\boldsymbol{W}_0^{(l)} + \boldsymbol{X}^{(l)}\boldsymbol{W}_1^{(l)}\right). \tag{3}$$

This form not only gains from the renormalization trick, but also makes the GCN more expressive and overcomes the over-smoothing issue, since the linear component in Eq. (3) retains a different value for each node even with infinitely many propagation layers.

3 A NEW BAG OF TRICKS

3.1 Label Usage

Label as Input. For semi-supervised classification tasks, apart from the graph G and the feature set $\boldsymbol{X}$, we also have access to the label set $\boldsymbol{Y}$, in which some nodes have missing labels and need to be predicted. However, prior work seldom considers the explicit use of label information in the training and inference procedures. Instead, the label information is usually regarded only as the target for supervised learning. However, when the training accuracy is below 100%, the label information of the misclassified samples is not contained in the model, though they can provide additional information. Additionally, samples misclassified by the model could potentially mislead their neighbors. Some approaches propose to address this problem by combining GNN with LPA, which can be

Algorithm 1: Label Usage for Graph Neural Networks

INPUT: G, $\boldsymbol{X}$, $\boldsymbol{Y}$, $f_{\boldsymbol{\theta}}$, the recycling times R

1: **for** each epoch **do**
2: Obtain $\mathcal{D}^L_{train}, \mathcal{D}^U_{train}$ by randomly splitting $\mathcal{D}_{train}$
3: $y_i^L \leftarrow \begin{cases} y_i, & (v_i, \boldsymbol{x}_i, y_i) \in \mathcal{D}^L_{train} \\ 0, & \text{otherwise} \end{cases}$
4: $\hat{\boldsymbol{Y}}^{(0)} \leftarrow f_{\boldsymbol{\theta}}(\boldsymbol{X} \parallel \boldsymbol{Y}^L, \boldsymbol{A})$
5: **for** $k \leftarrow 1$ to R **do**
6: $y_i^{(k-1)} \leftarrow \begin{cases} y_i, & (v_i, \boldsymbol{x}_i, y_i) \in \mathcal{D}^L_{train} \\ \hat{y}_i^{(k-1)}, & \text{otherwise} \end{cases}$
7: $\hat{\boldsymbol{Y}}^{(k)} \leftarrow f_{\boldsymbol{\theta}}(\boldsymbol{X} \parallel \boldsymbol{Y}^{(k-1)}, \boldsymbol{A})$
8: **end for**
9: Compute $\mathcal{L}(\hat{\boldsymbol{Y}}^{(R)}, \boldsymbol{Y})$ and update $\boldsymbol{\theta}$
10: **end for**

generally formulated as

$$\hat{\boldsymbol{Y}} = f_{LPA}(f_{GNN}(\boldsymbol{X}, \boldsymbol{A}), \boldsymbol{Y}), \tag{4}$$

where $\hat{\boldsymbol{Y}}$ is the label predictions and f_{LPA} and f_{GNN} denote the generic mapping functions of LPA and GNN, respectively. While these methods exploit the ground truth labels as part of the inference stage, the use of LPA for GNN post-processing may result in suboptimal solutions since f_{LPA} is a nonparametric function and f_{GNN} is a parametric one, and LPA relies on the assumption that adjacent nodes tend to share similar labels. To address this issue, we propose a novel masking technique independent of LPA that allows parametric GNN models to learn interrelationships between labels by taking label information as input, making it more suitable for heterogeneous and heterophily graphs.

Our method starts with a random split of $\mathcal{D}_{train}$ into several sub-datasets. For simplicity, we consider the case of two sub-datasets here, denoted as $\mathcal{D}^L_{train}$ and $\mathcal{D}^U_{train}$, respectively. Next, we mask the labels of $\mathcal{D}^U_{train}$, and learn to recover their labels. Specifically, the input for $\mathcal{D}^L_{train}$ contains both features and labels, while the input for $\mathcal{D}^U_{train}$ contains only features. and the labels used as inputs are set to zero-valued null vectors. During the final inference procedure, all labels in the training set are used as inputs to the model. We summarize this training procedure in Algorithm 1, where $f_{\boldsymbol{\theta}}$ denotes an arbitrary GNN model with parameters $\boldsymbol{\theta}$.

Augmentation with Label Reuse. We further propose *label reuse*, which recycles the predicted soft labels of the previous iteration and uses these soft labels as input. In this case, the labels of $\mathcal{D}^U_{train}$ and all test nodes are not assigned with null vectors but the prediction results of the previous iteration. In Algorithm 1, line 5 to line 8 presents the label reuse procedure.

3.2 Robust Loss Function for Classification

In binary classification scenarios, given feature space $\mathcal{X}$ and label space $\mathcal{Y} = \{-1, +1\}$, we aim to learn a classifier g that maps $\boldsymbol{x} \in \mathcal{X}$ to $\mathcal{Y}$. The classifier follows the decision rule $g(\boldsymbol{x}) = \text{sign}(f(\boldsymbol{x}))$ for some mapping f from $\mathcal{X}$ to $\mathbb{R}$. The optimization objective is to minimize the risk, defined as

$$R_\phi(f) = \mathbb{E}_{\mathcal{D}}[\ell(f(\boldsymbol{x}), y)] = \mathbb{E}_{\mathcal{D}}[\phi(yf(\boldsymbol{x}))], \tag{5}$$

where $\ell(f(\boldsymbol{x}), y)$ is the loss function and $\phi : \mathbb{R} \to \mathbb{R}^+$ is known as the *margin-based loss function*. Thus, choosing the loss function corresponds to choosing $\phi(\cdot)$.

A straightforward choice for $\phi(\cdot)$ is the *0-1 loss*

$$\ell_{0/1}(f(\boldsymbol{x}), y) = \phi_{0/1}(yf(\boldsymbol{x})) := H(-yf(\boldsymbol{x})), \tag{6}$$

where $H(\cdot)$ denotes the Heaviside step function. However, $\phi_{0/1}(\cdot)$ is a discontinuous function and is therefore computationally challenging to optimize. As a result, instead of directly optimizing *0-1 loss*, we turn to using $\phi(\cdot)$ as an upper bound of $\phi_{0/1}(\cdot)$, often referred to as the *calibrated surrogate loss*, for the optimization objective.

Being the most commonly used loss function for classification, the *logistic loss*, denoted as $\phi_{logit}(\cdot)$, provides a convex upper bound for $\phi_{0/1}(\cdot)$, which takes the form

$$\phi_{logit}(v) = \log(1 + \exp(-v)). \tag{7}$$

While the logistic loss performs satisfactorily in most cases, it suffers from sensitivity to outliers, whereas non-convex loss functions could be more robust [11]. Motivated by this, we consider weakening the convexity condition, and thereby designing a quasi-convex loss to contribute robustness:

$$\phi_{\rho-logit}(v) = \rho(\phi_{logit}(v)), \tag{8}$$

where $\rho : \mathbb{R}^+ \to \mathbb{R}^+$ is a non-decreasing function.

Some loss functions have been proposed to achieve outlier robustness, e.g., the *Savage loss* [11] and $\mathcal{L}_q$ loss [23], which can be interpreted as choosing a suitable $\rho(\cdot)$. Table 1 summarizes different $\rho(\cdot)$ of these and other loss functions discussed. As seen, both the *Savage loss* and the $\mathcal{L}_q$ loss have a finite upper bound, which causes some samples to be almost permanently ignored.

Here, we propose the *Loge loss* as a more preferable possibility for $\rho(\cdot)$:

$$\rho_{loge}(z) = \log(\epsilon + z) - \log\epsilon, \tag{9}$$

where ϵ is a tunable parameter and is fixed to $1 - \log 2$ throughout this paper so that $\frac{d^2}{dv^2}\ell_{loge}\Big|_{v=0} = 0$. This implies that the derivative of the Loge loss reaches its maximum at $z = 0$, which is exactly the decision boundary for classification tasks. Since the derivative of $\phi(\cdot)$ is the weight of the data sample [9], this may facilitate the optimization of accuracy.

Figure 1 illustrates the visualization of Loge loss and other losses along with their derivatives. As can be seen that Loge loss, Savage loss and $\mathcal{L}_q$ loss all reduce the weights of misclassified samples as v decreases, but loss functions other than Loge loss rapidly reduce to close to 0. In addition to this, it is seen that the derivative of Loge loss reaches its maximum at $v = 0$.

The Loge loss can be extended for multi-class classification tasks. We formulate the labels in a one-hot fashion, where both y and $\hat{y}$ are one-hot vectors, $\hat{y}_i$ denotes the value of the i-th element in $\hat{y}$, and the predicted value of the target class is denoted by $\hat{y}_{class}$ (i.e., the class-th element of y is 1). The Loge loss can then be formulated as

$$\ell_{loge}(\hat{y}, y) = \log\left(\epsilon - \log\frac{\exp(\hat{y}_{class})}{\sum_{i=1}^{C}\exp(\hat{y}_i)}\right) - \log\epsilon. \tag{10}$$

Loss	$\rho(z)$	$\rho(\phi_{logit}(v))$
Logistic	z	$\log(1+\exp(-v))$
Exponential	$\exp(z)-1$	$\exp(-v)$
Sigmoid	$1-\exp(-z)$	$\frac{1}{1+\exp(v)}$
Savage	$(1-\exp(-z))^2$	$\frac{1}{(1+\exp(v))^2}$
$\mathcal{L}_q$	$\frac{1}{q}(1-\exp(-qz))$	$\frac{1}{q}\left(1-\frac{1}{(1+\exp(-v))^q}\right)$
Loge	$\log(\epsilon+z)-\log\epsilon$	$\log(\epsilon+\log(1+\exp(-v)))-\log\epsilon$

Table 1: Loss functions with different $\rho(\cdot)$.

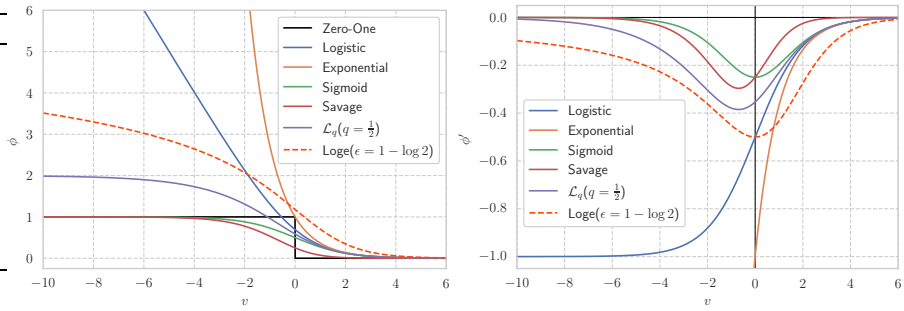

Figure 1: Visualization of various margin-based losses. The figure on the left and right illustrates ϕ and the derivative of ϕ, respectively.

3.3 Architecture Design

GAT with Symmetric Normalized Adjacency Matrix. As formulated in Eq. (2), the original message passing rule of GAT with self-loops can be written as

$$\boldsymbol{X}^{(l+1)} = \sigma\left(\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}}_{att}\boldsymbol{X}^{(l)}\boldsymbol{W}_0^{(l)} + \boldsymbol{X}^{(l)}\boldsymbol{W}_1^{(l)}\right), \tag{11}$$

where $\tilde{\boldsymbol{A}}_{att} = \boldsymbol{I}_N + \boldsymbol{A}_{att}$ is the normalized attention matrix, satisfying $\sum_i \tilde{\boldsymbol{A}}_{att,ij} = \sum_i \tilde{\boldsymbol{A}}_{ij}$ for each j.

Since the original GAT does not benefit from the the advantages of symmetric normalized adjacency matrix used in GCN, we propose a variant of GAT, with message passing formulation as

$$\boldsymbol{X}^{(l+1)} = \sigma\left(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}_{att}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{X}^{(l)}\boldsymbol{W}_0^{(l)} + \boldsymbol{X}^{(l)}\boldsymbol{W}_1^{(l)}\right). \tag{12}$$

Note that when $\boldsymbol{A}_{att} = \boldsymbol{A}$, Eq. (12) of GAT is equivalent to Eq. (3) of GCN.

Other GAT Variants. The attention mechanism of original GAT can be expressed as

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T[\boldsymbol{W}\boldsymbol{x}_i \parallel \boldsymbol{W}\boldsymbol{x}_j]\right)\right)}{\sum_{k\in\mathcal{N}(v_i)}\exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T[\boldsymbol{W}\boldsymbol{x}_i \parallel \boldsymbol{W}\boldsymbol{x}_k]\right)\right)}, \tag{13}$$

where $\boldsymbol{a}$ is a weight vector of the single-layer feed-forward neural network and $\parallel$ denotes the concatenation operation.

By replacing $\boldsymbol{a}^T\boldsymbol{W}$ with $\boldsymbol{a}^T$, Eq. (2) can be simplified to

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T[\boldsymbol{x}_i \parallel \boldsymbol{x}_j]\right)\right)}{\sum_{k\in\mathcal{N}(v_i)}\exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T[\boldsymbol{x}_i \parallel \boldsymbol{x}_k]\right)\right)}. \tag{14}$$

Another variant is the non-interactive GAT, which performs similarly to and at times better than the original form, and can be expressed as

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T\boldsymbol{x}_j\right)\right)}{\sum_{k\in\mathcal{N}(v_i)}\exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T\boldsymbol{x}_k\right)\right)}. \tag{15}$$

We also propose a GAT variant that exploits the edge features in the graph:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T[\boldsymbol{x}_i^V \parallel \boldsymbol{x}_j^V \parallel \boldsymbol{x}_{ij}^E]\right)\right)}{\sum_{k\in\mathcal{N}(v_i)}\exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T[\boldsymbol{x}_i^V \parallel \boldsymbol{x}_k^V \parallel \boldsymbol{x}_{ij}^E]\right)\right)}, \tag{16}$$

where $\boldsymbol{x}^V, \boldsymbol{x}^E$ denote node features and edge features, respectively. The time complexity of computing one layer of a single-headed GAT with C^V node features, C^E edge features and F filters is $O(|V|C^V F + |E|C^E + |E|F)$.

Table 2: Datasets statistics, where *label rate* denotes the proportion of labeled nodes used for training to the total nodes.

Dataset	#Nodes	#Edges	Metric	Label rate
Cora	2,708	5,429	Accuracy	5.2%
Citeseer	3,327	4,732	Accuracy	3.6%
Pubmed	1,9717	44,338	Accuracy	0.03%
Reddit	232,965	114,615,892	Accuracy	65.9%
Arxiv	169,343	1,166,243	Accuracy	53.7%
Proteins	132,534	39,561,252	ROC-AUC	65.4%
Products	2,449,029	61,859,140	Accuracy	8.0%

4 EXPERIMENTS

In this section, we examine the performance of each method though ablation experiments, and report the mean classification accuracy for multi-class classification tasks and Area Under the ROC Curve (ROC-AUC) for binary classification tasks. We choose three commonly used citation network datasets, namely Cora, Citeseer and Pubmed [15] and three relatively large datasets from OGB [4], namely ogbn-arxiv, ogbn-proteins and ogbn-products, plus Reddit, a dataset of posts from the Reddit website.[1] Some statistics of these datasets are presented in Table 2. Since ogbn-proteins is a dataset with edge features and ogbn-products is a huge dataset, we adopt neighbor sampling for them due to memory constraints. For Cora, Pubmed and Citeseer, we report the average scores and standard deviations after 100 runs, and for the relatively larger datasets ogbn-arxiv, ogbn-proteins, ogbn-products and Reddit, we report mean scores and standard deviations after 10 runs. All models and experiments were implemented using the Deep Graph Library (DGL) [19]. The results of baselines are comparable with their reports on OGB learderboard.

Label Usage. There are two principal factors that determine the benefits of the label usage in the training set. One is the proportion of the training set and the other is training accuracy. We investigate the performance of label as input and label reuse on datasets with a relatively large proportion of training set and low training accuracy. The results are reported in Table 3. Our approach improves the performance consistently with only a small increase in

[1] https://snap.stanford.edu/graphsage/

Table 3: Accuracy results (as measured by classification accuracy and ROC-AUC for ogbn-arxiv and ogbn-proteins, respectively) of different datasets and models in terms of label usage and GAT variant. *GCN+linear* indicates the GCN variant with a residual connection of Eq 3. *GAT indicates the GAT variant that incorporates the edge features.**

Dataset	Model	Label Usage	Accuracy(%)
Arxiv	GCN	–	72.48 ± 0.11
Arxiv	GCN	label as input	72.64 ± 0.10
Arxiv	GCN	label reuse	**72.78 ± 0.17**
Arxiv	GCN+linear	–	72.74 ± 0.13
Arxiv	GCN+linear	label as input	73.13 ± 0.14
Arxiv	GCN+linear	label reuse	**73.22 ± 0.13**
Arxiv	GAT	–	73.20 ± 0.16
Arxiv	GAT	label as input	73.24 ± 0.10
Arxiv	GAT	label reuse	**73.43 ± 0.13**
Arxiv	GAT(norm.adj.)	–	73.59 ± 0.14
Arxiv	GAT(norm.adj.)	label as input	73.66 ± 0.11
Arxiv	GAT(norm.adj.)	label reuse	73.91 ± 0.12
Arxiv	GAT(norm.adj.)	label reuse+C&S	**73.95 ± 0.12**
Arxiv	AGDN	–	73.75 ± 0.21
Arxiv	AGDN	label as input	**73.98 ± 0.09**
Proteins	GCN	–	80.07 ± 0.95
Proteins	GCN	label as input	**80.80 ± 0.56**
Proteins	GAT*	–	87.47 ± 0.16
Proteins	GAT*	label as input	**87.65 ± 0.08**

Table 4: Comparative results of logistic loss and Loge loss on different datasets and models, where ϵ of the Loge loss is $1 - \log 2$.

Dataset	Model	Accuracy(%)		
		Logistic	Savage	Loge
Cora	MLP	59.72 ± 1.01	**61.10 ± 0.91**	60.39 ± 0.74
Cora	GCN	82.26 ± 0.84	81.65 ± 0.74	**82.60 ± 0.83**
Citeseer	MLP	57.75 ± 1.05	**59.60 ± 0.92**	59.07 ± 0.98
Citeseer	GCN	71.13 ± 1.12	71.10 ± 1.22	**72.49 ± 1.12**
Pubmed	MLP	73.15 ± 0.68	**73.39 ± 0.62**	72.93 ± 0.65
Pubmed	GCN	78.89 ± 0.71	78.91 ± 0.63	**78.93 ± 0.69**
Reddit	MLP	72.98 ± 0.09	68.64 ± 0.29	**73.12 ± 0.09**
Reddit	GCN	**95.22 ± 0.04**	92.29 ± 0.48	95.18 ± 0.03
Arxiv	MLP	56.18 ± 0.14	51.97 ± 0.20	**56.72 ± 0.15**
Arxiv	GCN	71.77 ± 0.34	68.47 ± 0.32	**72.43 ± 0.16**
Arxiv	GAT	73.08 ± 0.26	69.58 ± 1.00	**73.20 ± 0.16**
Arxiv	GAT(norm.adj.)	73.29 ± 0.17	69.22 ± 1.48	**73.59 ± 0.14**
Products	MLP	62.90 ± 0.16	58.13 ± 1.03	**63.20 ± 0.13**
Products	GAT	80.99 ± 0.16	77.48 ± 0.14	**81.39 ± 0.14**

parameters. Furthermore, we can further improve the performance by combining our method with C&S [5].

Loss Function. We evaluate the performance of our loss function on datasets with classification accuracy as the metric. Results are reported in Table 4, where each model is trained with the same hyperparameters, varying only the loss functions. As shown, while robust Savage loss performs well on some small datasets, it performs considerably worse on larger datasets. Meanwhile, the Loge loss outperforms other losses on most datasets.

GAT Variant. To explore the effect of symmetric normalized adjacency matrix for GAT, we compare its performance with the original GAT on 5 datasets. The results are reported in Table 5. We see that GAT with normalized adjacency matrix achieves higher performance on all datasets. Nonetheless, we recommend choosing the appropriate adjacency matrix for different datasets. In Table 3, our GAT variant that incorporates the edge features *outperforms all other methods applied to the ogbn-proteins dataset on the OGB leaderboard at the time of this submission by a large margin.*

Table 5: Results of GAT variant. GAT+norm.adj. corresponds to GAT with symmetric normalized adjacency.

Dataset	Accuracy(%)	
	vanilla GAT	GAT+norm.adj.
Cora	83.41 ± 0.74	**83.72 ± 0.74**
Citeseer	71.92 ± 0.92	**72.25 ± 1.04**
Pubmed	78.43 ± 0.64	**78.77 ± 0.54**
Reddit	96.97 ± 0.04	**97.06 ± 0.05**
Arxiv	73.20 ± 0.16	**73.59 ± 0.14**

5 CONCLUSION AND FUTURE WORK

In this paper, we present a novel framework that takes labels as input, propose a robust loss function and investigate a dozen tricks for training deep GNNs with promising performance. These techniques can be applied to various GNN architectures, some of which only require minor modifications to data processing, loss function, or model architectures. Beyond our empirical results, it is worth exploring the label usage and loss functions from a theoretical perspective for further study.

REFERENCES

[1] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *arXiv:1801.10247*.
[2] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*.
[3] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *arXiv:1706.02216*.
[4] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. In *arXiv:2005.00687*.
[5] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R Benson. 2020. Combining Label Propagation and Simple Models Out-performs Graph Neural Networks. In *arXiv:2010.13993*.
[6] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. In *arXiv:1609.02907*.
[7] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. In *arXiv:1810.05997*.
[8] Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. 2020. Flag: Adversarial data augmentation for graph neural networks. In *arXiv:2010.09891*.
[9] Christian Leistner, Amir Saffari, Peter M Roth, and Horst Bischof. 2009. On robustness of on-line boosting-a competitive study. In *ICCV Workshops*.
[10] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. 2020. Deepergcn: All you need to train deeper gcns. In *arXiv:2006.07739*.
[11] Hamed Masnadi-shirazi and Nuno Vasconcelos. 2009. On the Design of Loss Functions for Classification: theory, robustness to outliers, and SavageBoost. In *NeurIPS*.
[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *arXiv:1310.4546*.
[13] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. Dropedge: Towards deep graph convolutional networks on node classification. In *arXiv:1907.10903*.
[14] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. 2020. Sign: Scalable inception graph neural networks. In *arXiv:2004.11198*.
[15] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. In *AI magazine*.
[16] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. 2020. Masked label prediction: Unified message passing model for semi-supervised classification. In *arXiv:2009.03509*.
[17] Chuxiong Sun and Guoshi Wu. 2020. Adaptive Graph Diffusion Networks with Hop-wise Attention. In *arXiv:2012.15024*.

[18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. In *arXiv:1710.10903*.
[19] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. In *arXiv:1909.01315*.
[20] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. 2020. Adversarial examples improve image recognition. In *CVPR*.
[21] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. 2020. Understanding negative sampling in graph representation learning. In *KDD*.
[22] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*.
[23] Zhilu Zhang and Mert R Sabuncu. 2018. Generalized cross entropy loss for training deep neural networks with noisy labels. In *arXiv:1805.07836*.
[24] Xiaojin Jerry Zhu. 2005. Semi-supervised learning literature survey.
[25] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *arXiv:1911.07323*.